Starte ROS-prosjekt

Hvordan starte sitt første ROS-prosjekt:

1. Lag en Catkin workspace (fullfør Tiago-turtorial først)

```
$ catkin_create_pkg navn_paa_prosjektet rospy roscpp
$ catkin build
$ cd <catkin_workspace>
$ source devel/setup.bash
```

    a. Denne kommandoen må legges under src i catkin workspace ditt, som i mitt tilfelle heter *tiago_public_ws*

2. Naviger frem til mappen *src* under *navn_paa_prosjektet* og opprett en python-fil, *publisher.py*
    a. I *publisher.py* ta med:

```
#! /usr/bin/env python
import rospy
from std_msgs.msg import String
#### CREATES A NEW NODE ####
# Node name
rospy.init_node('tiago_master')
#### CREATES A NEW TOPIC ####
# What should be published can be cosumized
#topic name
publisher = rospy.Publisher('/say_hello', String, queue_size=1)rate =
rospy.Rate(3) # 3Hz

while not rospy.is_shutdown():
   publisher.publish('Hey!')
   rate.sleep()
```

    b. For å kjøre koden over
        i. I en terminal skriv:

```
$ roscore
```

        ii. I en annen terminal skriv:

```
$ rosrun navn_paa_prosjektet publisher.py
```

    c. Kan være at du må gjøre filen kjørbar:

```
$ chmod +x navn_paa_prosjektet/src/publisher.py
```

    d. Du kan nå finne /say_hello topic under *rostopic*
    e. Kjør /say_hello slik:
        i. I en tredje terminal skriv:

```
$ rostopic echo /say_hello
```

/say_hello er også en node som kjøres. Du kan finne alle tilgengelige noder ved å skrive

```
$ rosnode list
```

For å kjøre /say_hello noden kan du også skrive

```
$ rosrun nann_paa_prosjektet say_hello
```

Og for å kjøre flere noder samtidig så må du bruke roslaunch

```
$ roslaunch openni_launch openni.launch
```

CMakeList.txt

Beskriver hvordan man skal kjøre programmet og hvor den skal installeres. CMakeList må følge disse funksjonene i riktig rekkefølge!

1. Required CMake Version (
   `cmake_minimum_required(VERSION 2.8.3)`
2. Package Name (
   `project(<navn_paa_prosjekt>)`
3. Find other CMake/Catkin packages needed for build
   `find_package(catkin REQUIRED)`
4. Enable Python module support (catkin_python_setup())
5. Message/Service/Action Generators (add_message_files(), add_service_files(), add_action_files())
6. Invoke message/service/action generation (generate_messages())
7. Specify package build info export
   ```
   catkin_package(
      INCLUDE_DIRS include
      LIBRARIES ${PROJECT_NAME}
      CATKIN_DEPENDS roscpp nodelet
      DEPENDS eigen opencv)
   ```
8. Libraries/Executables to build (add_library()/add_executable()/target_link_libraries())
9. Tests to build (catkin_add_gtest())
10. Install rules (install())

How to compile python code i in ROS

Trenger først en catkin-pakke (catkin_create_pkg). Kjøre catking_make for builde (eller catkin build) deretter source setup.bash i devel. For å kjøre skriv rosrun <pakagenavn> <pythonfilnavn>.py

ROS messager

To receive and send messages between ROS nodes.
Small example:

Publisher:

```
#! /usr/bin/env python
import rospy
from std_msgs.msg import String

rospy.init_node('msg_example_sender')
rate = rospy.Rate(3) # 3Hz
msg = String()
msg.data = "Sending a message from msg_example"
publisher = rospy.Publisher('/msg', String, queue_size=1)
while not rospy.is_shutdown():
   publisher.publish(msg)
   rate.sleep()
```

Subscriber:

```
#! /usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(msg):
      printf("Received msg: %s", msg.data)

rospy.init_node('msg_example_receiver')
subscriber = rospy.Subscriber('/msg', String, callback)
rospy.spin()
```

ROS messager kan ha forskjellige strukturer, i eksempelet over brukte jeg String. Men du kan også ha med flere parametere. Som for eksempel (twist inneholder linear- og angular-komponenter for x, y og z):

```
from std_msgs.msg import Twist, Vector3
message = Twist()
linear = Vector3()
linear.x = 1.0
```

```
linear.y = 2.0
linear.z = 3.0

angular = Vector3()
angular.x = 12.0
angular.y = 45.0
angular.z = 10.0

message.linear = linear
message.angular = angular
```

msg-folder

Det er også mulig å lage en msg-fil i en msg-folder inne i workspace/package ditt (catkin_create_pkg). Du kan opprette en slik fil ved å skrive dette i package-området:

```
$ mkdir msg
$ cd msg
$ touch message.msg
```

og i message.msg kan du for eksempel skrive:

```
string name
float32 number
```

I package.xml må du uncomment linjene:

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

og i CMakeList.txt må du legge til std_msgs og message_generation i find_package

```
find_package(catkin REQUIRED COMPONENTS
      rospy
      std_msgs
      message_generation
)
```

og uncomment add_message_files i CMakeList.txt

```
add_message_files(
      FILES
      message.msg
)
```

og uncomment generate_messages i CMakeList.txt

```
generate_messages(
      DEPENDENCIES
      std_msgs
)
```

og i catkin_package i CMakeList.txt må du legge til ‘DEPENDS message_runtime’

```
catkin_package(
      DEPENDS message_runtime
)
```

For å kompilere kan du gå ut til catkin workspace og skrive

```
$ catkin_make
$ # eller
$ catkin build
```

Deretter kan du sjekke om du har messagen ved å skrive (grep betyr bare at du sorterer ut alle msg som inneholder message i seg)

```
$ rosmsg list | grep message
```

For å sjekke innholdet i message.msg kan du skrive

```
$ rosmsg info <navn_paa_package>/message
```

ROS coordinates

Konvensjoner for koordinater. Alle systmer er 'høyre-hendte' (lat som tomel er z-aksen, pekefingeren er x-aksen og langefingeren er y-aksen).
I relasjon til for en 'body' så er standaren for linear orientasjon:

- x = forward
- y = left
- z = up

Rotasjonsorientasjonen følger en angular-systemet, så med høyrehåndsregelen vil en positiv z-verdi rotere counter clockwise rundt 'tommelen' din.